

```

// vsgdi_nt.c.txt
// Copyright (c) 2003. Zone Labs, Inc. All Rights Reserved.
/*****/
/**          ZoneLabs TrueVector Engine          **/
/**          Copyright(c) Zone Labs, Inc. 2003          **/
/*****/
/*
    vsgdi_nt.c (vsdatant.sys)
    Driver component for WinNT, hooking/monitoring of GDI calls
    History:
        GF 01/15/2003 Created
*/
#include "ntddk.h"
#include "stdarg.h"
#include "stdio.h"
#include "vsdatant.h"
#include "vserror.h"
#include "vsdriver.h"
#define WM_KEYFIRST          0x0100
#define WM_KEYDOWN           0x0100
#define WM_KEYUP             0x0101
#define WM_CHAR              0x0102
#define WM_DEADCHAR          0x0103
#define WM_SYSKEYDOWN        0x0104
#define WM_SYSKEYUP          0x0105
#define WM_SYSCHAR           0x0106
#define WM_SYSDEADCHAR       0x0107
#define WM_KEYLAST           0x0108
#define WM_COMMAND           0x0111
#define WM_SYSCOMMAND        0x0112
#define WM_TIMER             0x0113
#define WM_MOUSEFIRST        0x0200
#define WM_MOUSEMOVE         0x0200
#define WM_LBUTTONDOWN        0x0201
#define WM_LBUTTONUP         0x0202
#define WM_LBUTTONDBLCLK     0x0203
#define WM_RBUTTONDOWN        0x0204
#define WM_RBUTTONUP         0x0205
#define WM_RBUTTONDBLCLK     0x0206
#define WM_MBUTTONDOWN        0x0207
#define WM_MBUTTONUP         0x0208
#define WM_MBUTTONDBLCLK     0x0209
#define WM_MOUSEWHEEL         0x020A
#define WM_XBUTTONDOWN        0x020B
#define WM_XBUTTONUP         0x020C
#define WM_XBUTTONDBLCLK     0x020D
#define WM_MOUSELAST         0x020D
#define BM_SETCHECK          0x00F1
#define BM_GETSTATE          0x00F2
#define BM_SETSTATE          0x00F3

```

```

#define BM_SETSTYLE    0x00F4
#define BM_CLICK      0x00F5
#define BM_GETIMAGE    0x00F6
#define BM_SETIMAGE    0x00F7
// GDI call prototypes
#define USER_MESSAGE_CALL_SERVICE_NT 0x00000000
#define USER_MESSAGE_CALL_SERVICE_2K 0x000011bc
#define USER_MESSAGE_CALL_SERVICE_XP 0x000011cc
typedef NTSTATUS
(NTAPI
*NT_USER_MESSAGE_CALL) (
    HWND hWnd,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam,
    LRESULT lResult,
    DWORD dwUnknown1,
    DWORD dwUnknown2);
#define USER_POST_MESSAGE_SERVICE_NT 0x00000000
#define USER_POST_MESSAGE_SERVICE_2K 0x000011cb
#define USER_POST_MESSAGE_SERVICE_XP 0x000011db
typedef NTSTATUS
(NTAPI
*NT_USER_POST_MESSAGE) (
    HWND hWnd,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam);
#define USER_SEND_INPUT_SERVICE_NT 0x00000000
#define USER_SEND_INPUT_SERVICE_2K 0x000011e1
#define USER_SEND_INPUT_SERVICE_XP 0x000011f6
typedef UINT
(NTAPI
*NT_USER_SEND_INPUT) (
    UINT nInputs, // count of input events
    PVOID pInput, // struct _LPINPUT pInputs - array of input events
    int cbSize // size of structure
);
#define USER_QUERY_WINDOW_SERVICE_NT 0x00000000
#define USER_QUERY_WINDOW_SERVICE_2K 0x000011d2
#define USER_QUERY_WINDOW_SERVICE_XP 0x000011e3
#define QUERY_WINDOW_PROCESS 0x00000000
#define QUERY_WINDOW_THREAD 0x00000001
typedef DWORD
(NTAPI
*NT_USER_QUERY_WINDOW) (
    HWND hWnd,
    DWORD dwQuery); // See QUERY_WINDOW_???
#define USER_GET_FOREGROUND_WINDOW_SERVICE_NT 0x00000000
#define USER_GET_FOREGROUND_WINDOW_SERVICE_2K 0x00001189

```

```

#define USER_GET_FOREGROUND_WINDOW_SERVICE_XP 0x00001194
typedef HWND
(NTAPI
*NT_USER_GET_FOREGROUND_WINDOW) ();
NT_USER_MESSAGE_CALL UserMessageCallHandler = NULL;
NT_USER_POST_MESSAGE UserPostMessageHandler = NULL;
NT_USER_SEND_INPUT UserSendInputHandler = NULL;
HOOK_FUNCTION hUserMessageCall = {0};
HOOK_FUNCTION hUserPostMessage = {0};
HOOK_FUNCTION hUserSendInput = {0};
// Utility stuff we need to call
NT_USER_QUERY_WINDOW NtUserQueryWindow = NULL;
NT_USER_GET_FOREGROUND_WINDOW NtUserGetForegroundWindow = NULL;
#define MAX_PROCESS_FILTER 4
DWORD dwProcFilter[MAX_PROCESS_FILTER] = {0};
HWND hWndLast = NULL;
DWORD dwProcIdLast = 0;
BOOL IsProcessFilter(DWORD dwProcessID)
{
    DWORD n;
    for (n = 0; n < MAX_PROCESS_FILTER; n++)
    {
        if (dwProcFilter[n] == dwProcessID)
            return TRUE;
    }
    return FALSE;
}
NTSTATUS AddProcessFilter(DWORD dwProcessID)
{
    DWORD n;
    for (n = 0; n < MAX_PROCESS_FILTER; n++)
    {
        if (dwProcFilter[n] == dwProcessID)
            return STATUS_SUCCESS;
        if (dwProcFilter[n] == 0)
        {
            dwProcFilter[n] = dwProcessID;
            return STATUS_SUCCESS;
        }
    }
    return STATUS_UNSUCCESSFUL;
}
NTSTATUS DelProcessFilter(DWORD dwProcessID)
{
    DWORD n;
    for (n = 0; n < MAX_PROCESS_FILTER; n++)
    {
        if (dwProcFilter[n] == dwProcessID)
        {
            dwProcFilter[n] = 0;
        }
    }
}

```

```

    return STATUS_SUCCESS;
}
}
return STATUS_UNSUCCESSFUL;
}
BOOL WndContinue(
    HWND hWnd)
{
    if (NtUserQueryWindow)
    {
        DWORD dwProcessID;
        if (hWnd == hWndLast)
        {
            dwProcessID = dwProcIdLast;
        }
        else
        {
            dwProcessID = NtUserQueryWindow(hWnd, QUERY_WINDOW_PROCESS);
            dwProcIdLast = dwProcessID;
            hWndLast = hWnd;
        }
        if (IsProcessFilter(dwProcessID) &&
            (dwProcessID != GetCurrentProcessID()))
        {
            return FALSE;
        }
    }
    return TRUE;
}
BOOL MsgContinue(
    HWND hWnd,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam)
{
    BOOL bProtect = FALSE;
    BOOL bContinue = TRUE;
    switch (Msg)
    {
        // If we're not interested in a message at all, let's find that out first
        //case WM_QUIT:
        case WM_TIMER:
            if (lParam == 0)
                break;
        // fall through
        case WM_KEYDOWN:
        case WM_SYSKEYDOWN:
        case WM_KEYUP:
        case WM_SYSKEYUP:
        case WM_LBUTTONDOWN:

```

```

case WM_LBUTTONUP:
case WM_RBUTTONDOWN:
case WM_RBUTTONUP:
case WM_COMMAND:
case BM_SETSTATE:
case BM_SETCHECK:
case BM_CLICK:
case WM_MBUTTONDOWN:
case WM_MBUTTONUP:
case WM_MBUTTONDOWNBLCLK:
case WM_RBUTTONDOWNBLCLK:
case WM_LBUTTONDOWNBLCLK:
    return WndContinue(hWnd);
}
return TRUE;
}
NTSTATUS
NTAPI
HookUserMessageCall(
    HWND hWnd,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam,
    LRESULT lResult,
    DWORD dwUnknown1,
    DWORD dwUnknown2)
{
    NTSTATUS Status = STATUS_SUCCESS;
    if (MsgContinue(hWnd, Msg, wParam, lParam) && UserMessageCallHandler)
    {
        Status = UserMessageCallHandler(hWnd, Msg, wParam, lParam, lResult,
            dwUnknown1, dwUnknown2);
    }
#ifdef _DEBUG
    else
    {
        DbgPrint("[GDI NtUserMessageCall] BLOCKED: "
            "hWnd %x, Msg %x, wParam %x, lParam %x, "
            "Process %x\n",
            hWnd, Msg, wParam, lParam,
            GetCurrentProcessID());
    }
#endif // _DEBUG
    return Status;
}
NTSTATUS
NTAPI
HookUserPostMessage(
    HWND hWnd,
    UINT Msg,

```

```

WPARAM wParam,
LPARAM lParam)
{
    NTSTATUS Status = STATUS_SUCCESS;
    if (MsgContinue(hWnd, Msg, wParam, lParam) && UserPostMessageHandler)
    {
        Status = UserPostMessageHandler(hWnd, Msg, wParam, lParam);
    }
#ifdef _DEBUG
    else
    {
        DbgPrint("[GDI NtUserPostMessage] BLOCKED: "
            "hWnd %x, Msg %x, wParam %x, lParam %x, "
            "Process %x\n",
            hWnd, Msg, wParam, lParam,
            GetCurrentProcessID());
    }
#endif // _DEBUG
    return Status;
}

UINT
NTAPI
HookUserSendInput(
    UINT nInputs,    // count of input events
    PVOID pInputs,   // struct _LPINPUT pInputs - array of input events
    int cbSize)      // size of structure
{
    HWND hWnd = 0;
    if (NtUserGetForegroundWindow &&
        WndContinue(hWnd = NtUserGetForegroundWindow()) &&
        UserSendInputHandler)
    {
        return UserSendInputHandler(nInputs, pInputs, cbSize);
    }
#ifdef _DEBUG
    else
    {
        DbgPrint("[GDI NtUserSetInput] BLOCKED: "
            "hWnd %x, Process %x\n",
            hWnd, GetCurrentProcessID());
    }
#endif // _DEBUG
    return 0;
}

NTSTATUS StartTrackGDI()
{
    NTSTATUS Status = STATUS_SUCCESS;
    Status = HookInt2EService(
        &hUserMessageCall,
        HookUserMessageCall,

```

```

    FindInt2EServiceByID(USER_MESSAGE_CALL_SERVICE));
if (Status == STATUS_SUCCESS)
    UserMessageCallHandler = hUserMessageCall.pOldFunction;
Status = HookInt2EService(
    &hUserPostMessage,
    HookUserPostMessage,
    FindInt2EServiceByID(USER_POST_MESSAGE_SERVICE));
if (Status == STATUS_SUCCESS)
    UserPostMessageHandler = hUserPostMessage.pOldFunction;
Status = HookInt2EService(
    &hUserSendInput,
    HookUserSendInput,
    FindInt2EServiceByID(USER_SEND_INPUT_SERVICE));
if (Status == STATUS_SUCCESS)
    UserSendInputHandler = hUserSendInput.pOldFunction;
Status = FindInt2EServiceCall(
    FindInt2EServiceByID(USER_QUERY_WINDOW_SERVICE),
    (PVOID*)&NtUserQueryWindow));
Status = FindInt2EServiceCall(
    FindInt2EServiceByID(USER_GET_FOREGROUND_WINDOW_SERVICE),
    (PVOID*)&NtUserGetForegroundWindow));
return Status;
}

NTSTATUS StopTrackGDI()
{
    NTSTATUS Status = STATUS_SUCCESS;
    Status = UnhookInt2EService(&hUserMessageCall);
    Status = UnhookInt2EService(&hUserPostMessage);
    Status = UnhookInt2EService(&hUserSendInput);
    return Status;
}

// vslpc_nt.c.txt
// Copyright (c) 2003. Zone Labs, Inc. All Rights Reserved.
/*****
**          ZoneLabs TrueVector Engine          **
**          Copyright(c) Zone Labs, Inc. 2003          **
*****/
/*
    vslpc_nt.c (vsdatant.sys)
    Driver component for WinNT, hooking/monitoring of LPC calls
    History:
        GF 01/15/2003 Created
*/
#include "ntddk.h"
#include "stdarg.h"
#include "stdio.h"
#include "vsdatant.h"
#include "vserror.h"
#include "vsdriver.h"
#ifdef _DEBUG

```

```

#define _DEBUG_LPC
#endif
// Various data structures
typedef struct _LPC_SECTION_OWNER_MEMORY {
    ULONG          Length;
    HANDLE          SectionHandle;
    ULONG          OffsetInSection;
    ULONG          ViewSize;
    PVOID          ViewBase;
    PVOID          OtherSideViewBase;
} LPC_SECTION_OWNER_MEMORY, *PLPC_SECTION_OWNER_MEMORY;
typedef struct _LPC_SECTION_MEMORY {
    ULONG          Length;
    ULONG          ViewSize;
    PVOID          ViewBase;
} LPC_SECTION_MEMORY, *PLPC_SECTION_MEMORY;
typedef struct _LPC_MESSAGE {
    USHORT          DataLength;
    USHORT          Length;
    USHORT          MessageType;
    USHORT          DataInfoOffset;
    CLIENT_ID       ClientId;
    ULONG           MessageId;
    ULONG           CallbackId;
} LPC_MESSAGE, *PLPC_MESSAGE;
// LPC call prototypes
typedef NTSTATUS
(NTAPI
*NT_CONNECT_PORT) (
    OUT PHANDLE          ClientPortHandle,
    IN PUNICODE_STRING   ServerPortName,
    IN PSECURITY_QUALITY_OF_SERVICE SecurityQos,
    IN OUT PLPC_SECTION_OWNER_MEMORY ClientSharedMemory,
    OUT PLPC_SECTION_MEMORY ServerSharedMemory,
    OUT PULONG           MaximumMessageLength,
    IN OUT PVOID         ConnectionInfo,
    IN OUT PULONG        ConnectionInfoLength);
NTSYSAPI
NTSTATUS
NTAPI
NtConnectPort(
    OUT PHANDLE          ClientPortHandle,
    IN PUNICODE_STRING   ServerPortName,
    IN PSECURITY_QUALITY_OF_SERVICE SecurityQos,
    IN OUT PLPC_SECTION_OWNER_MEMORY ClientSharedMemory,
    OUT PLPC_SECTION_MEMORY ServerSharedMemory,
    OUT PULONG           MaximumMessageLength,
    IN OUT PVOID         ConnectionInfo,
    IN OUT PULONG        ConnectionInfoLength);
#endif _DEBUG_LPC

```



```

// GFNOTE: This API doesn't exists in NT. Most parameters are guesses at this point,
// I didn't find any prototypes online. So this breaks the driver for NT in debug
// mode ...
#define SECURE_CONNECT_PORT_SERVICE_NT 0x00000000
#define SECURE_CONNECT_PORT_SERVICE_2K 0x000000b8
#define SECURE_CONNECT_PORT_SERVICE_XP 0x000000d2
typedef NTSTATUS
(NTAPI
*NT_SECURE_CONNECT_PORT) (
    OUT PHANDLE      ClientPortHandle,
    IN PUNICODE_STRING  ServerPortName,
    IN PSECURITY_QUALITY_OF_SERVICE SecurityQos,
    IN OUT PLPC_SECTION_OWNER_MEMORY ClientSharedMemory,
    PVOID      pUnknown,
    OUT PLPC_SECTION_MEMORY ServerSharedMemory,
    OUT PULONG      MaximumMessageLength,
    IN OUT PVOID      ConnectionInfo,
    IN OUT PULONG      ConnectionInfoLength);
/* Not exported by ntoskrnl
NTSYSAPI
NTSTATUS
NTAPI
NtSecureConnectPort(
    OUT PHANDLE      ClientPortHandle,
    IN PUNICODE_STRING  ServerPortName,
    IN PSECURITY_QUALITY_OF_SERVICE SecurityQos,
    IN OUT PLPC_SECTION_OWNER_MEMORY ClientSharedMemory,
    PVOID      pUnknown,
    OUT PLPC_SECTION_MEMORY ServerSharedMemory,
    OUT PULONG      MaximumMessageLength,
    IN OUT PVOID      ConnectionInfo,
    IN OUT PULONG      ConnectionInfoLength);
*/
#define CREATE_PORT_SERVICE_NT 0x00000000
#define CREATE_PORT_SERVICE_2K 0x00000028
#define CREATE_PORT_SERVICE_XP 0x0000002e
typedef NTSTATUS
(NTAPI
*NT_CREATE_PORT) (
    OUT PHANDLE      PortHandle,
    IN POBJECT_ATTRIBUTES  ObjectAttributes,
    IN ULONG      MaxConnectInfoLength,
    IN ULONG      MaxDataLength,
    IN OUT PULONG      Reserved OPTIONAL );
/* Not exported by ntoskrnl
NTSYSAPI
NTSTATUS
NTAPI
NtCreatePort(
    OUT PHANDLE      PortHandle,

```

```

IN POBJECT_ATTRIBUTES ObjectAttributes,
IN ULONG             MaxConnectInfoLength,
IN ULONG             MaxDataLength,
IN OUT PULONG        Reserved OPTIONAL );
*/
#endif // _DEGUG_LPC
// Hook handlers and handles
NT_CONNECT_PORT ConnectPortHandler = NULL;
HOOK_FUNCTION hConnectPort = {0};
#ifdef _DEBUG_LPC
NT_SECURE_CONNECT_PORT SecureConnectPortHandler = NULL;
HOOK_FUNCTION hSecureConnectPort = {0};
NT_CREATE_PORT CreatePortHandler = NULL;
HOOK_FUNCTION hCreatePort = {0};
#endif // _DEBUG_LPC
NTSTATUS __stdcall OnProcessLpcDnsAccess(DWORD dwProcessID)
{
    // We are generating a "pseudo" WSock message here, not a process message
    PVMSG_STREAM pMsg;
    PHOOKREQUEST pHook = pWSockHook;
    NTSTATUS Status = STATUS_SUCCESS;
    if (dwProcessID == dwMonitorProcessID)
        return STATUS_SUCCESS;
    if (pHook)
    {
        pMsg = NewMessage(
            pHook,
            sizeof(VMSG_STREAM),
            GetCurrentProcessID(),
            GetCurrentThreadId(),
            0, 0, 0);
        if (pMsg)
        {
            pMsg->dwMsgFlags |= MFM_NEEDREPLY;
            pMsg->dwMsgLevel = MSG_LEVEL_INFO_LOW;
            Status = PutMessage(pHook, &pMsg,
                MCWSOCK_LPC_DNS_ACCESS_BEFORE,
                NULL, 0, 0);
            FreeMessage(pHook, (PBASEVMSG)pMsg);
            if (Status)
                Status = STATUS_ACCESS_DENIED;
        }
    }
    return Status;
}
NTSTATUS
NTAPI
HookConnectPort(
    OUT PHANDLE        ClientPortHandle,
    IN PUNICODE_STRING  ServerPortName,

```

```

    IN PSECURITY_QUALITY_OF_SERVICE SecurityQos,
    IN OUT PLPC_SECTION_OWNER_MEMORY ClientSharedMemory,
    OUT PLPC_SECTION_MEMORY ServerSharedMemory,
    OUT PULONG      MaximumMessageLength,
    IN OUT PVOID     ConnectionInfo,
    IN OUT PULONG    ConnectionInfoLength)
{
    NTSTATUS Status = STATUS_SUCCESS;
    CHAR cPortName[256] = "";
    DWORD dwLen = 0;
    DWORD dwProcessID = GetCurrentProcessID();
    // GFNOTE: Should we run the whole thing in native UNICODE strings?
    // Do we need an exception handler here in case of bad parameters?
    if (ServerPortName)
        dwLen = UnicodeStringToChar(*ServerPortName, cPortName, sizeof(cPortName));
    // DNS request to services.exe
    if (lstrcmpi(cPortName, "\\RPC Control\\DNSResolver") == 0)
    {
        Status = OnProcessLpcDnsAccess(dwProcessID);
    }
    // add more checks here
    if ((Status == STATUS_SUCCESS))
    {
        if (ConnectPortHandler)
        {
            Status = ConnectPortHandler(
                ClientPortHandle,
                ServerPortName,
                SecurityQos,
                ClientSharedMemory,
                ServerSharedMemory,
                MaximumMessageLength,
                ConnectionInfo,
                ConnectionInfoLength);
        }
        else
        {
            Status = NtConnectPort(
                ClientPortHandle,
                ServerPortName,
                SecurityQos,
                ClientSharedMemory,
                ServerSharedMemory,
                MaximumMessageLength,
                ConnectionInfo,
                ConnectionInfoLength);
        }
    }
}

#ifdef _DEBUG
if (Status == STATUS_SUCCESS)

```

```

DbgPrint("[LPC NtConnectPort] ""%s"" (%x) OK - Port %x\n",
    cPortName, dwProcessID, ClientPortHandle ? *ClientPortHandle : 0);
else
    DbgPrint("[LPC NtConnectPort] ""%s"" (%x) FAIL - Status %x\n",
        cPortName, dwProcessID, Status);
#endif // _DEBUG
return Status;
}
#ifdef _DEBUG_LPC
NTSTATUS
NTAPI
HookSecureConnectPort(
    OUT PHANDLE          ClientPortHandle,
    IN PUNICODE_STRING    ServerPortName,
    IN PSECURITY_QUALITY_OF_SERVICE SecurityQos,
    IN OUT PLPC_SECTION_OWNER_MEMORY ClientSharedMemory,
    PVOID                pUnknown,
    OUT PLPC_SECTION_MEMORY ServerSharedMemory,
    OUT PULONG           MaximumMessageLength,
    IN OUT PVOID         ConnectionInfo,
    IN OUT PULONG        ConnectionInfoLength)
{
    NTSTATUS Status = STATUS_SUCCESS;
    CHAR cPortName[256] = "";
    DWORD dwLen = 0;
    DWORD dwProcessID = GetCurrentProcessID();
    // GFNOTE: Should we run the whole thing in native UNICODE strings?
    // Do we need an exception handler here in case of bad parameters?
    if (ServerPortName)
        dwLen = UnicodeStringToChar(*ServerPortName, cPortName, sizeof(cPortName));
    // DNS request to services.exe
    if (lstrcmpi(cPortName, "\\RPC Control\\DNSResolver") == 0)
    {
        Status = OnProcessLpcDnsAccess(dwProcessID);
    }
    // add more checks here
    if ((Status == STATUS_SUCCESS))
    {
        if (ConnectPortHandler)
        {
            Status = SecureConnectPortHandler(
                ClientPortHandle,
                ServerPortName,
                SecurityQos,
                ClientSharedMemory,
                pUnknown,
                ServerSharedMemory,
                MaximumMessageLength,
                ConnectionInfo,
                ConnectionInfoLength);
        }
    }
}

```

```

}
else
{
/* Status = NtSecureConnectPort(
    ClientPortHandle,
    ServerPortName,
    SecurityQos,
    ClientSharedMemory,
    pUnknown,
    ServerSharedMemory,
    MaximumMessageLength,
    ConnectionInfo,
    ConnectionInfoLength);*/
    Status = STATUS_ACCESS_DENIED;
}
}
#ifdef _DEBUG
if (Status == STATUS_SUCCESS)
    DbgPrint("[LPC NtSecureConnectPort] ""%s"" (%x) OK - Port %x\n",
        cPortName, dwProcessID, ClientPortHandle ? *ClientPortHandle : 0);
else
    DbgPrint("[LPC NtSecureConnectPort] ""%s"" (%x) FAIL - Status %x\n",
        cPortName, dwProcessID, Status);
#endif // _DEBUG
return Status;
}
NTSTATUS
NTAPI
HookCreatePort(
    OUT PHANDLE          PortHandle,
    IN POBJECT_ATTRIBUTES ObjectAttributes,
    IN ULONG              MaxConnectInfoLength,
    IN ULONG              MaxDataLength,
    IN OUT PULONG         Reserved OPTIONAL )
{
    NTSTATUS Status = STATUS_SUCCESS;
    CHAR cPortName[256] = "";
    DWORD dwLen = 0;
    DWORD dwProcessID = GetCurrentProcessID();
    // GfNOTE: Should we run the whole thing in native UNICODE strings?
    // Do we need an exception handler here in case of bad parameters?
    if (ObjectAttributes && ObjectAttributes->ObjectName)
        dwLen = UnicodeStringToChar(*(ObjectAttributes->ObjectName),
            cPortName, sizeof(cPortName));
    // add more checks here
    if ((Status == STATUS_SUCCESS))
    {
        if (CreatePortHandler)
        {
            Status = CreatePortHandler(

```

```

    PortHandle,
    ObjectAttributes,
    MaxConnectInfoLength,
    MaxDataLength,
    Reserved);
}
else
{
    /*Status = NtCreatePort(
        PortHandle,
        ObjectAttributes,
        MaxConnectInfoLength,
        MaxDataLength,
        Reserved); */
    Status = STATUS_ACCESS_DENIED;
}
}
#ifdef _DEBUG
if (Status == STATUS_SUCCESS)
    DbgPrint("[LPC NtCreatePort] \"%s\" (%x) OK - Port %x\n",
        cPortName, dwProcessID, PortHandle ? *PortHandle : 0);
else
    DbgPrint("[LPC NtCreatePort] \"%s\" (%x) FAIL - Status %x\n",
        cPortName, dwProcessID, Status);
#endif // _DEBUG
return Status;
}
#endif // _DEBUG_LPC
NTSTATUS StartTrackLPC()
{
    NTSTATUS Status;
    Status = HookInt2EService(
        &hConnectPort,
        HookConnectPort,
        FindInt2EService(NtConnectPort, 0));
    if (Status == STATUS_SUCCESS)
        ConnectPortHandler = hConnectPort.pOldFunction;
    else
        Status = STATUS_UNSUCCESSFUL;
#ifdef _DEBUG_LPC
    Status = HookInt2EService(
        &hSecureConnectPort,
        HookSecureConnectPort,
        FindInt2EServiceByID(SECURE_CONNECT_PORT_SERVICE));
    if (Status == STATUS_SUCCESS)
        SecureConnectPortHandler = hSecureConnectPort.pOldFunction;
    else
        Status = STATUS_UNSUCCESSFUL;
    Status = HookInt2EService(
        &hCreatePort,

```

```
    HookCreatePort,  
    FindInt2EServiceByID(CREATE_PORT_SERVICE));  
if (Status == STATUS_SUCCESS)  
    CreatePortHandler = hCreatePort.pOldFunction;  
else  
    Status = STATUS_UNSUCCESSFUL;  
#endif //_DEGUG_LPC  
return Status;  
}  
NTSTATUS StopTrackLPC()  
{  
    NTSTATUS Status;  
    Status = UnhookInt2EService(&hConnectPort);  
#ifdef _DEBUG_LPC  
    Status = UnhookInt2EService(&hSecureConnectPort);  
    Status = UnhookInt2EService(&hCreatePort);  
#endif //_DEGUG_LPC  
return Status;  
}
```